Preprocessing by Program Specialization

Karoline Malmkjær and Olivier Danvy
Computer Science Department
Aarhus University *

December 1994

As is well-known, searching for a string in a given text can be linearized in the string by preprocessing the text. Most preprocessing algorithms have been devised either top-down by instantiating abstract properties on words (*e.g.*, Knuth's derivation of the Knuth, Morris, & Pratt linear string-matching algorithm from Cook's theorem [7]), or bottom-up by observing possible regularities in the structure of the text. The present work describes an alternative approach to preprocessing that uses properties of a simple two-arguments matching program to determine structures yielding linear matching. We mechanically deduce a one-argument linear matching program and its tree-like data structures by specializing the two-arguments program with respect to a text.

We consider the usual naïve and quadratic solution: a string occurs in a text if it is a prefix of one of the suffixes of the text.

We use program specialization, or *partial evaluation* [4, 5]: specializing a source program with respect to part of its input yields a specialized version of the source program that, given the rest of the input, yields the same result as the source program, given the whole input. The correctness and the computability of program specialization stem from Kleene's S^m_n-theorem [6].

The naïve quadratic program expects any string and any text and returns the position of that string in that text or -1. Specializing this source program with respect to the text "aba" automatically yields a residual program that expects a string and returns the position of this string in "aba" or -1. This residual program can be expressed graphically by representing a successful test against the character a as the transition $\xrightarrow{a}$, and where each node $\odot$ has two implicit transitions: a success one if the pattern ends there, and a failure if no character matches the explicit transitions. This yields the following tree, which makes it possible to match a string linearly.

*Ny Munkegade, DK-8000 Aarhus C, Denmark. E-mail: (karoline, danvy)@daimi.aau.dk
Supported by the DART project funded by The Danish Research Councils.

$$\begin{tikzcd}
& \odot \arrow[r, "b"] & \odot \arrow[r, "a"] & \odot \\
{} \arrow[r] & \odot \arrow[u, "a"] \arrow[r, "b"'] & \odot \arrow[r, "a"'] & \odot
\end{tikzcd}$$

This tree is similar to Weiner's prefix trees [8] (a.k.a. position trees [1]), but it contains the full suffixes, whereas prefix trees only contain prefix identifiers. Therefore this tree grows quadratically with the text, whereas prefix trees are only worst-case quadratic.

We can fold it by relaxing the requirement on the source program to return the position of a string in a text: we only want to know *whether* this string occurs in this text. Specializing the corresponding source program with respect to the text "aba" automatically yields a residual program that expects a string and determines whether this string occurs in "aba" by traversing the string linearly. In this case, further compaction is possible: the tree can be folded into a directed acyclic word graph [2].

$$\begin{tikzcd}
& \odot \arrow[dr, "b"] & & \\
{} \arrow[r] & \odot \arrow[u, "a"] \arrow[r, "b"'] & \odot \arrow[r, "a"'] & \odot
\end{tikzcd}$$

This approach generalizes the derivation of the Knuth, Morris, & Pratt linear string-matching program by partial evaluation of a naïve and quadradic string-matching program [3].

Acknowledgements

Originally, this work was partially supported by the Commission for Educational Exchange Between Denmark and the United States of America, the Danish Research Academy, and the American-Scandinavian Foundation.

The diagrams were drawn with Kristoffer Rose's XY-pic package.

References

[1] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, 1974.

[2] A. Blumer, J. Blumer, D. Haussler, A. Ehrenfeucht, M. T. Chen, and J. Seiferas. The smallest automaton recognizing the subwords of a text. *Theoretical Computer Science*, 40:31–55, 1985.

[3] Charles Consel and Olivier Danvy. Partial evaluation of pattern matching in strings. *Information Processing Letters*, 30(2):79–86, January 1989.

[4] Charles Consel and Olivier Danvy. Tutorial notes on partial evaluation. In Susan L. Graham, editor, *Proceedings of the Twentieth Annual ACM Symposium on Principles of Programming Languages*, pages 493–501, Charleston, South Carolina, January 1993. ACM Press.

[5] Neil D. Jones, Carsten K. Gomard, and Peter Sestoft. *Partial Evaluation and Automatic Program Generation*. Prentice-Hall International, 1993.

[6] Stephen C. Kleene. *Introduction to Metamathematics*. D. van Nostrand, Princeton, New Jersey, 1952.

[7] Donald E. Knuth, James H. Morris, and Vaughan R. Pratt. Fast pattern matching in strings. *SIAM Journal on Computing*, 6(2):323–350, 1977.

[8] Peter Weiner. Linear pattern matching algorithms. In *IEEE Symposium on Switching and Automata Theory*, pages 1–11, New York, 1973.